Numerical analysis

University of Luxembourg – Session 3

Numerical Integration - Exercises

Exercise 1. Elementary quadrature rules

We would like to compute numerically

$$I = \int_0^1 e^{-x^2} dx$$

using an elementary quadrature rule. Create a function "my_integral(f, a, b, method)" that returns the evaluation of the midpoint, trapezoidal and Simpson quadrature rules (weights and points) over an interval [a, b]

Test it on the proposed integral and compare their accuracy. The reference value is approximately 0.7468241328124269

Exercise 2. Method of Undetermined Coefficients

Using the Method of Undetermined Coefficients, which matches the integral of all monomials, find the 3-stage quadrature ($n = 3$) over the interval $[-1,1]$ with the points $x_1 = -1,\ x_2 = 0$ and $x_3 = 1$. Please comment

Exercise 3. Composite quadrature formula

Integrate the function $f(x) = sin(1/x)$ over J = [0.05, 1]. The exact value of the integral is about 0.5028396202159002, but you should find a numerical approximation to it.

Using a subdivision of the interval [0.05, 1] into K equally sized intervals JK, approximate the integral above using the

1. left-rectangle
2. midpoint rule
3. trapezoidal rule
4. Simpson rule

For each of these methods, generate a plot that shows the error between the exact integral value and your numerical approximation as a function of the number of function evaluations you need. Verify that the orders of convergence of the different rules correspond to the ones you expect (even order).

For each of the methods, find how many function evaluations you need to achieve an accuracy of 10−6. Which method do you recommend?

Exercise 4. Gaussian rules

Now also add the following Gaussian rules

5. 2-point Gauss rule
6. 3-point Gauss rule

Using the built-in function from scipy.integrate import fixed_quad. Also evaluate the cost of the approach (number of function estimates) and comment.

Exercise 5. Improper integrals

Consider the improper integral

$$I = \int_0^{\pi/2} \frac{cos(x)}{\sqrt{x}} dx$$

Plot it using Python and try to integrate it using np.trapz(y, x) which is a composite trapezoidal rule implementation. What do you notice?

Then try to find a change of variable to “cancel” out the singularity. Then determine the integral value again. To compute a reference solution you can use an adaptive quadrature (from scipy.integrate import quad) with a tolerance close to epsilon machine.

Exercise 6. Monte Carlo example

The volume V_d of a d-dimensional sphere (hypersphere) with radius R is given by:

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)} R^d$$

Where Γ is the function that generalizes the factorial.

Estimate the volume of a 10-dimensional sphere with unit radius using the Monte Carlo method. How does the error change if you increase or decrease the number of random points n? How might the dimensionality d affect the accuracy of Monte Carlo integration?

It is recommended to use

- np.random.uniform for the points generation
- np.linalg.norm for the distance to origin evaluation
- from scipy.special import gamma for the reference solution

Exercise 7. Numerical resolution of a boundary value problem

We consider the following differential equation with homogeneous Dirichlet boundary conditions

$$\begin{cases} -y''(x) + p(x)y'(x) + q(x)y(x) = f(x), & \forall\, x \in]0,1[, \\ y(0) = y(1) = 0. \end{cases}$$

where f, p, q are some given continuous functions on]0, 1[. We want to get an approximation y^h of the solution y. To this end, we discretize the interval [0, 1] with $(n + 2)$ equidistant points x_k.

$$x_k = \frac{k}{n+1} = kh, \quad \forall\ k \in \{0, n+1\}$$

We have seen during the course that the first and second-order derivatives of y at x_k can be approximated by the following centered formulae

$$y'(x_k) \simeq \frac{y_{k+1} - y_{k-1}}{2h}, \quad y''(x_k) \simeq \frac{y_{k-1} - 2y_k + y_{k+1}}{h^2}$$

1. For any $k\ \in\ \{1, n\}$, we have

   $$-y''(x_k) + p^h(x_k)y'(x_k) + q^h(x_k)y(x_k) = f^h(x_k)$$

   Write a discrete equation valid for any k ∈ {0, n + 1} obtained by replacing $y(x_k)$ and its derivatives by their approximations. Do not forget the boundary conditions.

2. Deduce that the vector y_h is the solution to a linear system, where you will specify its size and explicitly write the matrix A_h

3. Write a function

```
def build_FDM_matrix(ph, qh, ...):

...

return Ah
```

that builds this matrix. Take a look at the documentation of scipy.sparse.diags to create a sparse matrix from diagonal blocks.

4. Write a script for your solver. You may follow the steps below

   (a) define a row vector corresponding to the discretization of the interval [0, 1]

   (b) define vectors corresponding to the data p_{h}, q_h and f_h

   (c) construct the matrix A_h

   (d) solve of the linear system with a built-in routine

   (e) compare the numerical result (start with a small grid) with the exact solution.

Let us remark that it is difficult to compare the solution obtained to the exact solution since this last solution is generally not known. This is the reason why, for testing, we choose an exact solution, some functions p and q, and then calculate the function f by using these functions. The functions p, q and f being known, we will be able to get an approximation y_h of y and compare them.

5. Choose simple functions p, q, and compute the function f such that $y_{\text{ex}}(x) = sin(\pi x)$. Test your solver and compare the numerical and exact solution qualitatively.

6. We define e_h as the approximation error

$$e_h = max_{1 \le k \le n} |y_{\text{ex}}(k) - y_k|.$$

Check that $e_h \to 0$ when $h \to 0$. Plot the convergence rate in a log-scale. What do you expect?